ARL-TN-0864 ● JAN 2018

US Army Research Laboratory

Measuring and Inferring the State of the User via the Microsoft Kinect with Application to Cyber Security Research

by Christopher J Garneau

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-TN-0864● JAN 2018

US Army Research Laboratory

Measuring and Inferring the State of the User via the Microsoft Kinect with Application to Cyber Security Research

by Christopher J Garneau
Human Research and Engineering Directorate, ARL

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
January 2018	Technical Note	Jul 2016 – Sep 2017

4. TITLE AND SUBTITLE
Measuring and Inferring the State of the User via the Microsoft Kinect with Application to Cyber Security Research

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Christopher J Garneau

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
US Army Research Laboratory
ATTN: RDRL-HRA-AA
Aberdeen Proving Ground, MD 21005-5066

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TN-0864

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Within the domain of cyber security research, there exists a need to better understand the methods and processes by which cyber security analysts perform various tasks and develop new analysis techniques and tools for these tasks where appropriate. This report provides a review of relevant literature for measuring and inferring the state of the user via the Microsoft Kinect within the broad domains of usability and/or psychology. Of the various studies and applications sampled, the use of the Kinect to measure the attention of a user sitting at a computer workstation emerged as particularly relevant given that cyber security network analysts perform computing tasks that require sustained attention over a period of time. Custom software for collecting metrics that may be useful for measuring attention using the Kinect has also been developed and is documented here.

15. SUBJECT TERMS
Microsoft Kinect, user state, attention, engagement, vigilance, cyber security, network analysts

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Christopher J Garneau
Unclassified	Unclassified	Unclassified	UU	34	19b. TELEPHONE NUMBER (Include area code) 410-278-5814

Contents

List of Figures

1. Introduction

In the design and evaluation of products and interfaces (including graphical user interfaces or tools), it is often instructive to assess the quality of the user's interaction. An assessment of "quality" might include metrics of performance, satisfaction, learning, or other measures depending on the nature of the interface/task. The assessment might use subjective evaluation techniques (e.g., questionnaires or focus group feedback) or objective evaluation techniques (e.g., measurement of time on tasks or advanced sensors that measure parameters like electrodermal response). An assessment often uses both objective and subjective metrics and almost always relies on more than one type of measurement to obtain insight into the nature of the user-interface interaction.

Traditionally, obtaining objective metrics beyond basic parameters like time and accuracy on a task would require expensive and somewhat invasive laboratory equipment such as an electroencephalography (EEG) device (e.g., Jenkins et al. [2009] presents a comparison of EEG, thermal imaging, and subjective questionnaires to assess affective experience for product design). However, the development of the Microsoft Kinect offers an opportunity to readily obtain additional assessments of the state of the user. As an instrument for research, the Kinect offers many advantages in that it is noninvasive/noncontact, accessible, straightforward to program to obtain useful results, well documented and well supported, and very affordable (Smisek et al. 2013).

Within the domain of cyber security research, there exists a need to better understand the methods and processes by which cyber security analysts perform various tasks and develop new analysis techniques and tools for these tasks where appropriate. For instance, visualization tools might lessen analyst workload and improve situation awareness for analysts monitoring network intrusion systems (e.g., see recent research by the author: Garneau et al. [2016b, 2016a]). It is imperative to subsequently evaluate any new approaches with cyber security analysts, preferably in an operational environment or an environment that very closely mirrors the operational environment. Such conditions dictate that any instrumentation used in an evaluation be transparent to the analyst. Interruptions to analyst workflow (e.g., to assess situation awareness) should be avoided and the use of contact sensors would be uncomfortable or impractical for the long duration of a typical analyst's shift.

Considering the human as part of a cyber security system is a relatively new development in the cyber security research domain; there exist few examples of novel techniques to measure the state of the user in these contexts. Therefore, this

report presents examples of how the Microsoft Kinect has been used to measure and infer the state of the user within the broad domains of usability and/or psychology research and how this research is situated within other applications of the Kinect. Following this review, subsequent discussion provides an approach for measuring attention for research in the cyber security domain via custom data-collection software.

2. Description of the Kinect Hardware and Software

The Microsoft Kinect is a hardware sensor that incorporates an IR laser emitter/projector, cameras capturing visible light (red, green, blue [RGB]) and IR/depth information, and one or more microphones. The fused image with both the RGB and depth components is referred to as RGB-depth (RGB-D); most discussions of the Kinect for research focus on this component of the Kinect (the depth camera) and so this report does not discuss use of the microphones. The Kinect connects to a computer via a USB interface; due to hardware and processing requirements, a single Kinect is typically paired with a single laptop. Another integral part of the Kinect is the software accompanying the physical hardware that gathers the various inputs, analyzes them, and exposes various parameters to the programmer/ researcher via an application programming interface (API).

Microsoft provides a robust software development kit (SDK), Kinect for Windows, which facilitates application of available APIs for the Kinect. The Kinect for Windows SDK offers capabilities like skeleton tracking, facial tracking, and speech recognition; other tools for working with the Kinect include OpenNI and OpenKinect (Cruz et al. 2012). The Development Cognitive Neuroscience (DCN) Lab at Indiana University provides an excellent online workshop/tutorial for programming with the Kinect for XBox One (the latest version of the sensor). The tutorial covers everything from procuring the sensor to configuring the development environment to writing sample applications using skeletal and face tracking (DCN Lab 2017).

3. Applications of the Microsoft Kinect for Research

While originally developed as an add-on for the Microsoft XBox gaming console, the Kinect has found many applications in research in a wide variety of domains. Use of the Kinect has been investigated for real-time modeling of indoor environments—with application, for instance, to robotics (Du et al. 2011; Henry et al. 2012); for hand tracking and natural user interfaces (Frati and Prattichizzo 2011; Ren et al. 2011); for object and human activity identification (Janoch et al. 2013; Ni et al. 2013); for whole-body pose, posture, and segment-length estimation

(Robinson and Parkinson 2013; Shotton et al. 2013; Shum et al. 2013); and for gait measurement (Stone and Skubic 2013; Pfister et al. 2014). These are only a few examples of the many applications of the Kinect for research. Han et al. (2013) discuss the technical details of how the Kinect is implemented as a computer vision sensor in many of these domains and how processing techniques incorporating the RGB-D camera offer distinct advantages over techniques using only a standard RGB camera. Roscoe et al. (2012) present some considerations on the viability of using the Kinect as a research tool and how it compares to other types of 3-D information-gathering systems.

Of interest to the current discussion is application of the Kinect for measuring the state of the user interacting with a system or interface (i.e., application of the Kinect for measuring parameters like attention, engagement, frustration, and so on). With this in mind, 3 application areas of interest emerged from the literature: 1) facial expression and emotion recognition, 2) gaze tracking and attention, and 3) sensing of other nonverbal cues.

3.1 Facial Expression and Emotion Recognition

The first application area of interest to the discussion at hand to emerge from the literature is the use of the Kinect to ascertain emotion based on facial expression. This application lies within the broader domain of "affective computing". There exist many techniques and algorithms by which facial expression may be ascertained via plain (RGB) images of a person's face that do not require the Kinect. However, Malawski et al. (2014) indicate that inclusion of the Kinect sensor may help facial expression recognition in poor lighting or nonfrontal head poses, despite the lower resolution of facial markers provided by the Face Tracking SDK compared with other image-based techniques (the second-generation Face API offers greater resolution). Any use of the Kinect to ascertain emotion from facial expression would require prior research that correlates facial cues with emotion; Szwoch (2014) reviews several available databases that aim to do this, emphasizing the databases that specifically include the RGB-D data captured by the Kinect.

Some research has investigated use of the Kinect to develop an automated system for providing feedback to the user of a computer system based on their affect. Patwardhan and Knapp (2016) present a system (EmoFit) that uses key logging, activity interruptions, eye tracking, facial expression tracking, body posture, and speech to gauge a user's affect with the goal of monitoring the user during sedentary jobs (e.g., computer programming for the study). The EmoFit system also provided predictions of affect and a "health booster" based on the user's affect. Mar Saneiro and Boticario (2014) describe use of the Kinect to tag facial expression and body

movements corresponding with changes in affective states of learners while dealing with various cognitive tasks in an e-learning environment. In addition to the Kinect input, the authors use expert codification of emotion to build an automated affective support model to be provided to the user during learning tasks.

In addition to the previous study, other studies in affective computing using the Kinect have come from the learning domain. Grafsgaard et al. (2013) present research on automatically recognizing facial indicators of frustration during learning; certain action units within the Facial Action Coding System (FACS) were studied and the Computer Expression Recognition Toolbox (CERT) provided automatic coding to create models of facial expression, frustration, and learning. Specifically, the study found statistical relationships between an "outer brow raise" and learning, "brow lowering" and frustration, and "mouth dimpling" and both frustration and learning. Lee et al. (2015) use upper body posture as assessed by the Kinect to automatically recognize engagement in children performing a multiple intelligence test on a computer; engagement was defined at 2 levels: "high" and "low".

3.2 Gaze Tracking and Attention

Several studies have made use of the Kinect to measure a user's gaze. Mora and Odobez (2012) estimate gaze under free-head movements. The approach first creates a 3-D face model to track the head under a variety of poses, and then generates eye images with respect to the head to determine gaze parameters, and then transforms the gaze parameters back to the estimated head pose to determine overall gaze. Li et al. (2014) present a similar methodology where gaze is measured by a high-definition (HD) webcam and the Kinect. Gaze is separated into local motion (driven by pupil movement) and global motion (driven by head movement). Given the appropriate correlative models, gaze may be used to estimate the attention and focus of users.

Stanley (2013) presents thesis work on the prediction of user attention using the Kinect. User experimentation yielded a statistical correlation of attention with certain measures of body posture (joint tracking) and head characteristics. Participants completed sustained attention tasks while the Kinect measured features of the user's body and head posture. Instruments employed to assess attention were the Psychology Experiment Building Language (PEBL) Continuous Performance Test (PCPT), Test of Attentional Vigilance (TOAV), and the PEBL Perceptual Vigilance Task (PPVT). In addition to quantitative statistical correlations, useful qualitative observations based on the data include 1) users looking away from the screen (head yaw greater than 15°) exhibited worse performance on the attention

tests and 2) users sitting closer to the screen exhibited better performance on the attention tests.

3.3 Sensing of Other Nonverbal Cues

Some researchers have used various Kinect parameters and techniques not mentioned in the prior 2 sections to measure nonverbal behaviors. Burba et al. (2012) describe approaches for estimating respiratory rate (by measuring the average depth of the user's chest point cloud) and "leg jiggling" (by measuring oscillations of the top of the knee as measured by the skeletal tracking and depth map). The authors' work is motivated by the presupposition that measuring these behaviors will improve interaction with virtual human agents. Frauendorfer et al. (2014) describe use of a "smart room" to perform "nonverbal social sensing" to automatically record and extract nonverbal (vocal and visual) cues in social interactions. The authors present a study wherein participants act as job applicants and professional recruiters evaluate videotapes of a job interview to make a hiring decision; the nonverbal cues best predicting the hiring decision were found to be average turn duration, increased tempo variation, and maintenance of eye contact.

4. Data-Collection Software Useful for Research in the Cyber Security Domain

Humans interface with computing systems in a variety of ways and investigating how the human-in-the-loop affects cyber security may be approached from various angles. The particular angle of interest in this discussion—as alluded to in the Introduction—is the task wherein cyber security network analysts sit at a computer in an operational environment monitoring intrusion detection systems to discriminate between actual intrusion attempts and false alarms. This is a task that requires vigilance and stamina. As such, determining how well a particular approach or set of tools for monitoring cyber security networks enables the analyst to maintain attention for an extended period of time is an important consideration. Thus, of all the applications discussed in Section 3, using the Kinect to measure attention might be the best starting point for including the Kinect as a novel sensing tool for cyber security research.

Given the studies discussed in Section 3.2, Stanley (2013) provides a starting point for measuring attention using the Kinect, highlighting head and posture variables that are relevant to user attention (head yaw and upper body lean were noted as having greater correlation with measures of attention than many other variables that were assessed). While this work does not provide substantive, generalizable models that predict attention, it does indicate a general approach for measuring and

predicting the state of the user via the Microsoft Kinect. The following sections present and discuss custom software called AttentionGrabber. This is software that has been developed by the author to collect measures of interest that may be useful for developing models that correlate measures of attention with performance during extended intrusion detection activities. Such models may provide useful insight that may ultimately be used to better train analysts and develop better tools for their use.

4.1 Software Requirements

AttentionGrabber is implemented as a Windows Presentation Foundation (WPF) application that uses the Microsoft Kinect sensor (version 1). As such, the following are required to successfully install and run the executable:

- PC running Windows Vista or greater
- First-generation Kinect sensor with model number 1414, 1473, or 1517
- Microsoft .NET Framework
- Kinect for Windows Runtime v1.8

4.2 Interface Elements

The extensible application markup language (XAML) code defining the presentation of the application as well as the code-behind C# (CS) file may be found in the Appendix. Figure 1 shows the interface upon application startup (the application is configured as a single, fixed size, minimizable window called MainWindow.xaml).

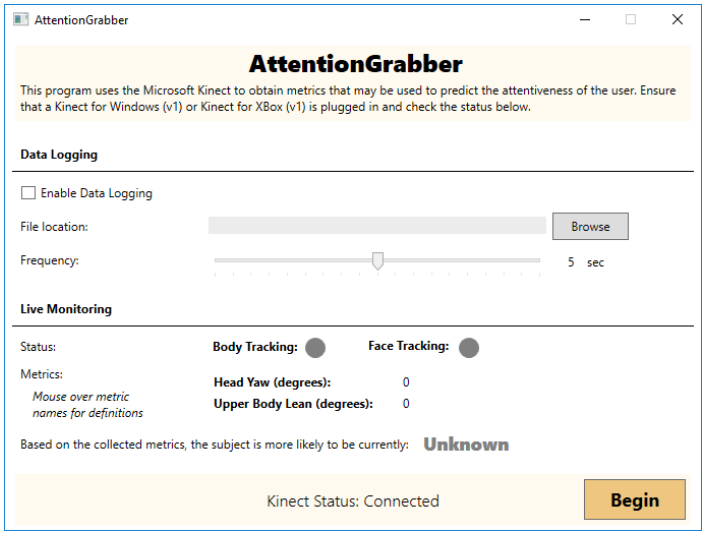

Fig. 1 AttentionGrabber interface on startup

AttentionGrabber will determine the status of a connected Kinect sensor upon startup and returns this status in the bar along the bottom of the application window. To be tracked, the user must be located at least 0.8 m (31.5 inches) from the sensor for Kinect models 1414 and 1473 or 0.4 m (15.7 inches) for Kinect model 1517 (Microsoft n.d.).

Figure 2 shows AttentionGrabber during data collection. There are several components of the interface that are described next.

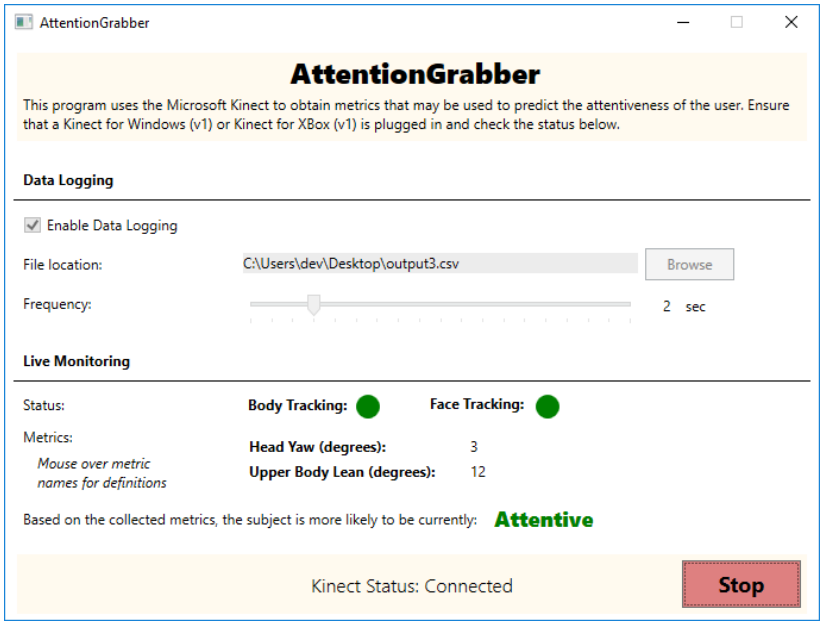

Fig. 2 AttentionGrabber interface during data collection

4.2.1 Data Logging Options

When the "Enable Data Logging" toggle is checked, the application will log data according to the file location and sampling frequency specified. Results are stored in a text/comma-separated value (CSV) file with a timestamp for each measurement. In Fig. 2, the application has been configured to log collected data to the "output3.csv" file on the desktop and sample measurements every 2 s. The sampling interval varies from 0.5 to 10 s. Figure 3 shows example output.

```
Time,Head_Yaw_Status,Head_Yaw,Upper_Body_Lean_Status,Upper_Body_Lean,Attention_Prediction
06:10:20.0,Not Tracked,0,Not Tracked,0,Unknown
06:10:21.3,Tracked,1,Tracked,-9,Inattentive
06:10:22.3,Tracked,4,Tracked,-10,Inattentive
06:10:23.3,Tracked,3,Tracked,-4,Inattentive
06:10:24.4,Tracked,0,Tracked,-10,Inattentive
06:10:25.7,Tracked,0,Tracked,-17,Inattentive
06:10:26.7,Tracked,28,Tracked,-17,Inattentive
06:10:27.8,Tracked,20,Tracked,-17,Inattentive
06:10:28.8,Tracked,3,Tracked,-17,Inattentive
06:10:29.8,Tracked,-18,Tracked,-16,Inattentive
06:10:30.9,Tracked,-10,Tracked,-16,Inattentive
06:10:31.9,Tracked,-3,Tracked,-3,Inattentive
06:10:33.2,Tracked,-3,Tracked,16,Attentive
06:10:34.2,Tracked,1,Tracked,25,Attentive
```

Fig. 3 Example output with a sampling interval of 1 s

4.2.2 Live Monitoring

Once the user clicks the "Begin" button, this section provides a continuously updated estimate of head yaw and upper body lean (both in degrees) along with tracking status (either "tracked" or "not tracked" as indicated by green or red circles). The metrics—suggested by Stanley (2013) for the measurement of attention—are defined as follows:

- **Head yaw:** Orientation (in degrees) of head about the vertical axis; an angle of 0 indicates that the user is looking head-on at the sensor and negative angles indicate clockwise rotation.
- **Upper body lean:** Angle (in degrees) of the upper body (defined by the hip and shoulder vertices) with respect to the vertical; negative angles indicate recline.

The application will also return a crude prediction of the user's likely attentive status as determined by comparisons for each of the metrics; if the absolute value of the user's head yaw is greater than 15° or the upper body lean is negative (recline), the user is determined to be more likely to be inattentive than attentive. This prediction is updated in real time.

4.2.3 Kinect Status Bar

This bar shows the status of the connected Kinect device along with the "Begin"/ "Stop" button. The possible Kinect statuses are unavailable/undefined, connected, disconnected, error, not ready, not powered, or initializing. For any status except "connected", a small icon and popup enable the user to select the Kinect sensor or view any problems or error messages.

4.3 Discussion and Limitations of the Current Software Implementation

AttentionGrabber has been configured to measure and return 2 performance metrics (head yaw and upper body lean); however, the application is easily extensible to measure many other metrics. Both skeleton tracking and the Face Tracking SDK have been implemented in the application; skeleton tracking enables live tracking of 20 joints and various custom derived measures, and the Face Tracking SDK enables live tracking of 100 points, 3 predefined derived measures, and many other custom derived measures (DCN Lab 2017). During a data collection activity, AttentionGrabber is intended to be minimized and run in the background with data logging enabled.

AttentionGrabber uses the first-generation Kinect sensor and not the newer second-generation sensor. In general, applications using the Kinect tend to be resource-intensive. This is particularly true for applications designed for the second-generation Kinect and this is one reason that the current software is implemented using a first-generation device. Second-generation devices also require a USB 3 connection that is less prevalent than the USB 2 connection required by the first-generation Kinect. A significant disadvantage of using a first-generation device for this particular purpose is the lower resolution compared with the second-generation Kinect. The second-generation device provides significantly greater resolution of facial features in particular and also provides a measure of eye gaze direction. Future implementations of AttentionGrabber could investigate use of the second-generation device.

5. Conclusion and Future Work

The literature review and data-collection software described in this report provide a foundation for future research to better understand the state of cyber security network analysts as they perform their duties. By relating physiological metrics obtained by the Kinect with analyst performance with various tasks or tools, follow-on work may yield predictive models of attention and objectively compare the effectiveness of different tasks/tools. Ultimately, this may yield better tools or training for analysts.

Future work should use the AttentionGrabber software to correlate the attentive state of the analyst with their performance. The Cyber Integrated Modeling and Experimentation Range – Army (CHIMERA) laboratory in the US Army Research Laboratory, Human Research and Engineering Directorate may be particularly well suited to apply this software to research involving human subjects. It may be helpful

to add additional metrics to the output of the software, which should be trivial given the framework already in place in the application. After preliminary experimentation using the prototype, future work might also investigate use of the second-generation Kinect sensor and the additional parameters that are returned as well as any computing performance degradation that may result from the greater requirements of this hardware.

6. References

Burba N, Bolas M, Krum DM, Suma EA. Unobtrusive measurement of subtle nonverbal behaviors with the Microsoft Kinect. In 2012 IEEE Virtual Reality Workshops (VRW), p. 1–4. IEEE, 2012.

Cruz L, Lucio D, Velho L. Kinect and RGBD images: Challenges and applications. In Graphics, Patterns and Images Tutorials (SIBGRAPI-T), 2012 25th SIBGRAPI Conference on, p. 36–49. IEEE, 2012.

Du H, Henry P, Ren X, Cheng M, Goldman DB, Seitz SM, Fox D. Interactive 3D modeling of indoor environments with a consumer depth camera. In Proceedings of the 13th International Conference on Ubiquitous Computing, p. 75–84. ACM, 2011.

Frati V, Prattichizzo D. Using Kinect for hand tracking and rendering in wearable haptics. In World Haptics Conference (WHC), 2011 IEEE, p. 317–321. IEEE, 2011.

Frauendorfer D, Mast MS, Nguyen L, Gatica-Perez D. Nonverbal social sensing in action: Unobtrusive recording and extracting of nonverbal behavior in social interactions illustrated with a research example. J Nonverbal Behavior. 2014;38(2):231–245.

Garneau C, Erbacher R, Etoty R. Evaluation of visualization tools for computer network defense analysts: Display design, methods, and results for a user study. Aberdeen Proving Ground (MD): Army Research Laboratory (US); 2016a. Report No.: ARL-TR-7869.

Garneau CJ, Erbacher RF, Etoty RE, Hutchinson SE. Results and lessons learned from a user study of display effectiveness with experienced cyber security network analysts. In Proceedings of the 2016 Learning from Authoritative Security Experiment Results (LASER) Workshop; 2016b; San Jose, CA.

Grafsgaard JF, Wiggins JB, Boyer KE, Wiebe EN, Lester JC. Automatically recognizing facial indicators of frustration: a learning-centric analysis. In Affective Computing and Intelligent Interaction (ACII), 2013 Humaine Association Conference on, p. 159–165. IEEE, 2013.

Han J, Shao L, Xu D, Shotton J. (2013). Enhanced computer vision with Microsoft Kinect Sensor: a review. IEEE Trans Cybernetics 43 [accessed 2017]. https://www.microsoft.com/en-us/research/publication/enhanced-computer-vision-with-microsoft-kinect-sensor-a-review/.

Henry P, Krainin M, Herbst E, Ren X, Fox D. RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments. The International Journal of Robotics Research. 2012;31(5):647–663.

Indiana University Developmental Cognitive Neuroscience (DCN) Lab. The Kinect for XBox One Workshop [accessed 2017]. http://www.indiana.edu/~dcnlab/KinectWorkshop/index.html.

Janoch A, Karayev S, Jia Y, Barron JT, Fritz M, Saenko K, Darrell T. A category-level 3D object dataset: Putting the Kinect to work. In Consumer Depth Cameras for Computer Vision, p. 141–165. Springer, 2013.

Jenkins S, Brown R, Rutterford N. Comparing thermographic, EEG, and subjective measures of affective experience during simulated product interactions. International Journal of Design. 2009;3(2).

Lee D, Han Y, Kyu W, Park C, Yoon H, Kim J, Park, C. Measuring the engagement level of children for multiple intelligence test using Kinect. In Seventh International Conference on Machine Vision (ICMV 2014), p. 944529–944529. International Society for Optics and Photonics, 2015.

Li Y, Monaghan DS, O'Connor NE. Real-time gaze estimation using a Kinect and a HD webcam. In International Conference on Multimedia Modeling. p. 506–517. Springer, 2014.

Malawski F, Kwolek B, Sako S. Using Kinect for facial expression recognition under varying poses and illumination. In International Conference on Active Media Technology, p. 395–406. Springer, 2014.

Mar Saneiro Olga C, Santos SS-M, Boticario JG. Towards emotion detection in educational scenarios from facial expressions and body movements through multimodal approaches, Scientific World J. 2014. doi: http://dx.doi.org/10.1155/2014/484873.

[Microsoft] Depth ranges. Redman (WA): Microsoft; n.d. [accessed 2017 Sep 1]. https://msdn.microsoft.com/en-us/library/hh973078.aspx#Depth_Ranges.

Mora KAF, Odobez J-M. Gaze estimation from multimodal Kinect data. In 2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, p. 25–30. IEEE, 2012.

Ni B, Wang G, Moulin P. RGBD-HUDAACT: A color-depth video database for human daily activity recognition. In Consumer Depth Cameras for Computer Vision, Berlin (Germany): Springer; 2013. p. 193–208.

Patwardhan A, Knapp G. EmoFit: affect monitoring system for sedentary jobs. arXiv preprint arXiv:1607.01077, 2016.

Pfister A, West AM, Bronner S, Noah JA. Comparative abilities of Microsoft Kinect and Vicon 3D motion capture for gait analysis. Journal of Medical Engineering & Technology. 2014;38(5):274–280.

Ren Z, Meng J, Yuan J, Zhang Z. Robust hand gesture recognition with Kinect sensor. In Proceedings of the 19th ACM International Conference on Multimedia, p. 759–760. ACM, 2011.

Robinson M, Parkinson M. Estimating anthropometry with Microsoft Kinect. In Proceedings of the 2nd International Digital Human Modeling Symposium, 2013.

Roscoe MS, Plöger PG, Kent KB, Herpers R. Determining the viability of the Kinect as a research tool. Fredericton (Canada): University of New Brunswick; 2012 Jan 23. Report No.: TR12-215.

Shotton J, Sharp T, Kipman A, Fitzgibbon A, Finocchio M, Blake A, Cook M, Moore R. Real-time human pose recognition in parts from single depth images. Communications of the ACM. 2013;56(1):116–124.

Shum HP, Ho ES, Jiang Y, Takagi S. Real-time posture reconstruction for Microsoft Kinect. IEEE Transactions on Cybernetics. 2013;43(5):1357–1369.

Smisek J, Jancosek M, Pajdla T. 3D with Kinect. In Consumer Depth Cameras for Computer Vision. Berlin (Germany): Springer; 2013. p. 3–25.

Stanley D. Measuring attention using Microsoft Kinect [master's thesis]. [Rochester (NY)]: Rochester Institute of Technology; 2013.

Stone EE, Skubic M. Unobtrusive, continuous, in-home gait measurement using the Microsoft Kinect. IEEE Transactions on Biomedical Engineering. 2013; 60(10):2925–2932.

Szwoch M. On facial expressions and emotions RGB-D database. In International Conference: Beyond Databases, Architectures and Structures. Berlin (Germany): Springer; 2014. p. 384–394.

INTENTIONALLY LEFT BLANK.

Appendix. Code for AttentionGrabber Software

This appendix appears in its original form, without editorial change.

A.1 XAML Presentation Code (MainWindow.xaml)

```
<Window x:Class="KinectAttention.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
xmlns:k="http://schemas.microsoft.com/kinect/2013"
xmlns:local="clr-namespace:KinectAttention"
mc:Ignorable="d"
Title="AttentionGrabber" Height="525" Width="700"
ResizeMode="CanMinimize">
<Grid>

<!-- Header block -->
<Rectangle Fill="#FFFFFAEF" HorizontalAlignment="Left"
Height="74" Margin="10,10,10,0" VerticalAlignment="Top"
Width="680"/>
<Label x:Name="label_logo" Content="AttentionGrabber"
HorizontalAlignment="Center" HorizontalContentAlignment="Center"
Margin="233,5,228,0" VerticalAlignment="Top" FontSize="24"
FontFamily="Segoe UI Black" Width="233" />
<TextBlock x:Name="text_description" HorizontalAlignment="Left"
Margin="15,45,15,0" TextWrapping="Wrap" Text="This program uses
the Microsoft Kinect to obtain metrics that may be used to
predict the attentiveness of the user. Ensure that a Kinect for
Windows (v1) or Kinect for XBox (v1) is plugged in and check the
status below. " VerticalAlignment="Top" Width="667"/>

<!-- Data logging options block -->
<Label x:Name="label_dataLogging" Content="Data Logging"
HorizontalAlignment="Left" Height="27" Margin="10,103,0,0"
VerticalAlignment="Top" Width="97" FontWeight="Bold"/>
<Rectangle Fill="Black" HorizontalAlignment="Center" Height="1"
Margin="7,133,7,0" Stroke="Black" VerticalAlignment="Top"
Width="680"/>
<CheckBox x:Name="checkBox_enableDataLogging" Content="Enable
Data Logging" HorizontalAlignment="Left" Margin="15,147,0,0"
VerticalAlignment="Top" Checked="checkBox_Checked"/>
<Label x:Name="label_fileLocation" Content="File location:"
HorizontalAlignment="Left" Height="27" Margin="10,174,0,0"
VerticalAlignment="Top" Width="123"/>
<TextBlock x:Name="parameter_fileLocation" Text=""
HorizontalAlignment="Left" Height="17" Margin="200,178,0,0"
VerticalAlignment="Top" Width="332" Background="#FFECECEC"/>
<Button x:Name="button_fileBrowse" Content="Browse"
Click="fileBrowse" HorizontalAlignment="Left"
Margin="538,174,0,0" VerticalAlignment="Top" Width="75"
Height="27"/>
<Label x:Name="label_sampleFrequency" Content="Frequency:"
HorizontalAlignment="Left" Height="27" Margin="10,207,0,0"
VerticalAlignment="Top" Width="123"
RenderTransformOrigin="0.968,0.512"/>
```

```
<Slider x:Name="slider_frequency" HorizontalAlignment="Left"
Height="30" Margin="201,213,0,0" VerticalAlignment="Top"
Width="330" Minimum="0.5" Maximum="9.5" Value="5"
TickFrequency="0.5" TickPlacement="BottomRight"
IsSnapToTickEnabled="True"/>
<Label x:Name="parameter_sampleFrequency" Content="{Binding
Value, ElementName=slider_frequency,
UpdateSourceTrigger=PropertyChanged}" HorizontalAlignment="Left"
Height="27" Margin="548,209,0,0" VerticalAlignment="Top"
Width="30" />
<Label x:Name="parameter_sampleFrequencyUnits" Content="sec"
HorizontalAlignment="Left" Height="27" Margin="567,209,0,0"
VerticalAlignment="Top" Width="50" />

<!-- Live monitoring output block -->
<Label x:Name="label_status" Content="Status:"
HorizontalAlignment="Left" Height="27" Margin="10,292,0,0"
VerticalAlignment="Top" Width="123"/>
<Label x:Name="label_bodyTracking" Content="Body Tracking:"
HorizontalAlignment="Left" Height="27" Margin="199,292,0,0"
VerticalAlignment="Top" Width="119" FontWeight="Bold" />
<Label x:Name="label_faceTracking" Content="Face Tracking:"
HorizontalAlignment="Left" Height="27" Margin="352,291,0,0"
VerticalAlignment="Top" Width="119" FontWeight="Bold" />
<Ellipse x:Name="bodyStatus" Fill="Gray"
HorizontalAlignment="Left" Height="20" Margin="295,297,0,0"
VerticalAlignment="Top" Width="20"/>
<Ellipse x:Name="faceStatus" Fill="Gray"
HorizontalAlignment="Left" Height="20" Margin="446,297,0,0"
VerticalAlignment="Top" Width="20"/>
<Label x:Name="label_liveMonitoring" Content="Live Monitoring"
HorizontalAlignment="Left" Height="27" Margin="10,255,0,0"
VerticalAlignment="Top" Width="123" FontWeight="Bold"/>
<Rectangle Fill="Black" HorizontalAlignment="Center" Height="1"
Margin="7,285,7,0" Stroke="Black" VerticalAlignment="Top"
Width="680"/>
<Label x:Name="label_metrics" Content="Metrics:"
HorizontalAlignment="Left" Height="27" Margin="10,319,0,0"
VerticalAlignment="Top" Width="123"/>
<Label x:Name="label_metricTooltips" Content="Mouse over
metric&#xA;names for definitions" HorizontalAlignment="Left"
Height="47" Margin="22,341,0,0" VerticalAlignment="Top"
Width="123" FontStyle="Italic"/>
<Label x:Name="label_headYaw" Content="Head Yaw (degrees): "
HorizontalAlignment="Left" Height="27" Margin="200,327,0,0"
VerticalAlignment="Top" Width="174" FontWeight="Bold"
ToolTip="Orientation of head about the vertical axis; an angle of
0 indicates that the user is looking head-on at the sensor and
negative angles indicate clockwise rotation"/>
<TextBlock x:Name="parameter_headYaw" Text="0"
HorizontalAlignment="Left" VerticalAlignment="Top" Height="27"
Margin="391,332,0,0" Width="123"/>
<Label x:Name="label_upperBodyLean" Content="Upper Body Lean
(degrees):" HorizontalAlignment="Left" Height="27"
Margin="200,348,0,0" VerticalAlignment="Top" Width="174"
FontWeight="Bold" ToolTip="Angle of the upper body (defined by
```

```
the hip and shoulder vertices); negative angles indicate
recline"/>
<TextBlock x:Name="parameter_upperBodyLean" Text="0"
HorizontalAlignment="Left" VerticalAlignment="Top" Height="27"
Margin="391,353,0,0" Width="123"/>
<Label x:Name="label_attentionStatus" Content="Based on the
collected metrics, the subject is more likely to be currently:"
HorizontalAlignment="Left" Height="27" Margin="10,388,0,0"
VerticalAlignment="Top" Width="404"/>
<Label x:Name="parameter_attentionStatus" Content="Unknown"
HorizontalAlignment="Center" Height="30" Margin="411,384,137,0"
VerticalAlignment="Top" FontSize="18" FontWeight="Bold"
FontFamily="Segoe UI Black" Foreground="Gray" Width="146" />

<!-- Kinect status bar -->
<Label x:Name="parameter_kinectStatus" Content="Kinect Status:
Unavailable" HorizontalAlignment="Center"
HorizontalContentAlignment="Center"
VerticalContentAlignment="Center" Margin="10,0,10,5"
VerticalAlignment="Bottom" FontSize="16" FontFamily="Segoe UI
Semilight" Height="50" Width="680" Background="#FFFFFAEF"/>
<k:KinectSensorChooserUI x:Name="kinectChooser"
HorizontalAlignment="Left" VerticalAlignment="Bottom"
Margin="15,0,0,10"/>
<Button x:Name="button_begin" Content="Begin"
Click="beginCollect" HorizontalAlignment="Right" Height="40"
Margin="0,0,15,10" VerticalAlignment="Bottom" Width="100"
FontWeight="Bold" FontSize="18" Background="#FFEEC680"/>
<TextBox x:Name="bodyStatusOutput" HorizontalAlignment="Left"
Height="23" Margin="548,297,0,0" TextWrapping="Wrap" Text="Not
Tracked" VerticalAlignment="Top" Width="120"
Visibility="Hidden"/>
<TextBox x:Name="faceStatusOutput" HorizontalAlignment="Left"
Height="23" Margin="548,327,0,0" TextWrapping="Wrap" Text="Not
Tracked" VerticalAlignment="Top" Width="120" Visibility="Hidden"
/>

</Grid>
</Window>
```

A.2 CS Code-Behind (MainWindow.xaml.cs)

```
using Microsoft.Kinect;
using Microsoft.Kinect.Toolkit;
using Microsoft.Kinect.Toolkit.FaceTracking;
using System;
using System.Linq;
using System.Windows;
using System.Windows.Input;
using System.Windows.Media;
using System.IO;
using System.Windows.Threading;

namespace KinectAttention
{
```

```
/// <summary>
/// Interaction logic for MainWindow.xaml
/// </summary>
///

public partial class MainWindow : Window
{
private FaceTracker faceTracker;
private byte[] colorImage;
private short[] depthImage;
private        ColorImageFormat        colorImageFormat        =
ColorImageFormat.Undefined;
private        DepthImageFormat        depthImageFormat        =
DepthImageFormat.Undefined;
public DispatcherTimer dispatcherTimer;

private KinectSensor sensor;
public MainWindow()
{
      InitializeComponent();
      Loaded += MainWindowLoaded;

      button_begin.Content = "Begin"; // force label since this is
used for status checking
}

private void MainWindowLoaded(object sender, RoutedEventArgs e)
{
      var sensorStatus = new KinectSensorChooser();

      sensorStatus.KinectChanged                                    +=
KinectSensorChooserKinectChanged;

      kinectChooser.KinectSensorChooser = sensorStatus;
      sensorStatus.Start();
}

private void checkBox_Checked(object sender, RoutedEventArgs e)
{
      // this is required, so don't delete it!
}

private   void   KinectSensorChooserKinectChanged(object   sender,
KinectChangedEventArgs e)
{

      if (sensor != null)
            sensor.AllFramesReady -= KinectAllFramesReady;

      sensor = e.NewSensor;

      if (sensor == null)
            return;

      switch (Convert.ToString(e.NewSensor.Status))
      {
```

```
            case "Connected":
                  parameter_kinectStatus.Content = "Kinect Status:
Connected";
                  break;
            case "Disconnected":
                  parameter_kinectStatus.Content = "Kinect Status:
Disconnected";
                  break;
            case "Error":
                  parameter_kinectStatus.Content = "Kinect Status:
Error";
                  break;
            case "NotReady":
                  parameter_kinectStatus.Content = "Kinect Status:
Not Ready";
                  break;
            case "NotPowered":
                  parameter_kinectStatus.Content = "Kinect Status:
Not Powered";
                  break;
            case "Initializing":
                  parameter_kinectStatus.Content = "Kinect Status:
Initialising";
                  break;
            default:
                  parameter_kinectStatus.Content = "Kinect Status:
Undefined";
                  break;
      }
}

private       void       KinectAllFramesReady(object       sender,
AllFramesReadyEventArgs e)
{
      // Executes each time data for a new frame (skeleton, depth,
and color) are ready

      // SKELETON TRACKING...

      //    helpful    to    understand    the    skeleton    space:
https://msdn.microsoft.com/en-us/library/hh973078.aspx
      var skeletons = new Skeleton[0];

      using (var skeletonFrame = e.OpenSkeletonFrame())
      {
            if (skeletonFrame != null)
            {
                  skeletons                  =                  new
Skeleton[skeletonFrame.SkeletonArrayLength];
                  skeletonFrame.CopySkeletonDataTo(skeletons);
            }
      }

      if (skeletons.Length == 0){
            return;
      }
```

```
      var skel = skeletons.FirstOrDefault(x => x.TrackingState ==
SkeletonTrackingState.Tracked);

      // body tracking status indicator (red/green)...
      if (skel == null)
      {
            bodyStatus.Fill = new SolidColorBrush(Colors.Red);
            bodyStatusOutput.Text = "Not Tracked";
            return;
      } else {
            bodyStatus.Fill = new SolidColorBrush(Colors.Green);
            bodyStatusOutput.Text = "Tracked";
      }


      double upperBodyLeanAngle = getUpperBodyLeanAngle(skel);
      parameter_upperBodyLean.Text                                    =
upperBodyLeanAngle.ToString();

      // FACE TRACKING...

      if (this.faceTracker == null)
      {
            try
            {
                  this.faceTracker = new FaceTracker(sensor);
            }
            catch (InvalidOperationException)
            {
                  // During some shutdown scenarios the FaceTracker
                  //  is  unable  to  be  instantiated.   Catch  that
exception
                  // and don't track a face.
                  this.faceTracker = null;
            }
      }

      ColorImageFrame colorImageFrame = null;
      DepthImageFrame depthImageFrame = null;

      colorImageFrame = e.OpenColorImageFrame();
      depthImageFrame = e.OpenDepthImageFrame();

      if (colorImageFrame == null || depthImageFrame == null)
      {
            return;
      }

      // Check for image format changes.  The FaceTracker doesn't
      // deal with that so we need to reset.
      if (this.depthImageFormat != depthImageFrame.Format)
      {
            this.depthImage = null;
            this.depthImageFormat = depthImageFrame.Format;
      }
```

```
      if (this.colorImageFormat != colorImageFrame.Format)
      {
            this.colorImage = null;
            this.colorImageFormat = colorImageFrame.Format;
      }

      // Create any buffers to store copies of the data we work
with
      if (this.depthImage == null)
      {
            this.depthImage                 =                 new
short[depthImageFrame.PixelDataLength];
      }

      if (this.colorImage == null)
      {
            this.colorImage                 =                 new
byte[colorImageFrame.PixelDataLength];
      }

      colorImageFrame.CopyPixelDataTo(this.colorImage);
      depthImageFrame.CopyPixelDataTo(this.depthImage);

      FaceTrackFrame faceFrame = faceTracker.Track(
      colorImageFormat, colorImage,
      depthImageFormat, depthImage, skel);

      // face tracking status indicator (red/green)...
      if (faceFrame.TrackSuccessful) {
            faceStatus.Fill = new SolidColorBrush(Colors.Green);
            faceStatusOutput.Text = "Tracked";
      } else {
            faceStatus.Fill = new SolidColorBrush(Colors.Red);
            faceStatusOutput.Text = "Not Tracked";
      }

      double headYaw = Math.Round(faceFrame.Rotation.Y);
      parameter_headYaw.Text = headYaw.ToString();

      // comparisons for attentiveness prediction
      if ((upperBodyLeanAngle>0) && (Math.Abs(headYaw)<15)){
            parameter_attentionStatus.Content = "Attentive";
            parameter_attentionStatus.Foreground       =       new
SolidColorBrush(Colors.Green);
      } else {
            parameter_attentionStatus.Content = "Inattentive";
            parameter_attentionStatus.Foreground       =       new
SolidColorBrush(Colors.Orange);
      }
}

private void saveData(object sender, EventArgs e)
{
      // save parameters of current frame to file when called
```

```
      string outputString = DateTime.Now.ToString("hh:mm:ss.f") +
"," + faceStatusOutput.Text +","+parameter_headYaw.Text + "," +
bodyStatusOutput.Text  +","+parameter_upperBodyLean.Text  +  ","+
parameter_attentionStatus.Content+Environment.NewLine;
      File.AppendAllText(parameter_fileLocation.Text,
outputString);

      // Forcing the CommandManager to raise the RequerySuggested
event
      CommandManager.InvalidateRequerySuggested();
}



private void fileBrowse(object sender, RoutedEventArgs e)
{
      // Create OpenFileDialog
      Microsoft.Win32.SaveFileDialog       dlg       =       new
Microsoft.Win32.SaveFileDialog();

      // Set filter for file name, file extension and default file
extension
      dlg.FileName = "output";
      dlg.DefaultExt = ".csv";
      dlg.Filter = "CSV files (*.csv)|*.csv";

      // Display OpenFileDialog by calling ShowDialog method
      Nullable<bool> result = dlg.ShowDialog();

      // Get the selected file name and display in a TextBox
      if (result == true)
      {
            // Open document
            string filename = dlg.FileName;
            parameter_fileLocation.Text = filename;
      }
}

private void beginCollect(object sender, RoutedEventArgs e)
{
      // executed when user clicks "Begin" button

      if     ((checkBox_enableDataLogging.IsChecked.Value)     &&
(parameter_fileLocation.Text == ""))
      {
            MessageBox.Show("You  have  selected  'Enable  Data
Logging' but have not specified a filename. Go back and click
'Browse' to select a location on your computer to save the log
file.", "AttentionGrabber Error");
      }
      else if (parameter_kinectStatus.Content != "Kinect Status:
Connected")
      {
            MessageBox.Show("Kinect  is  not  ready.  Remedy  the
problem before beginning.", "AttentionGrabber Error");
      }
```

```
      else
      {
            if (button_begin.Content == "Begin")
            {
                  // start data collection/streaming

                  sensor.SkeletonStream.Enable();
                  sensor.DepthStream.Enable();
                  sensor.ColorStream.Enable();

                  if  (checkBox_enableDataLogging.IsChecked.Value)
{
                        // Start timer
                        // helpful: https://msdn.microsoft.com/en-
us/library/system.windows.threading.dispatchertimer.aspx
                        //        might        be        helpful:
http://stackoverflow.com/questions/29382194/save-data-stream-to-
a-file-every-second-in-net

                        // write header row...

      File.WriteAllText(parameter_fileLocation.Text,
"Time,Head_Yaw_Status,Head_Yaw,Upper_Body_Lean_Status,Upper_Body_
Lean,Attention_Prediction" + Environment.NewLine);

                        int             saveInterval             =
(int)(slider_frequency.Value   *   1000);   //convert   seconds   to
milliseconds
                        dispatcherTimer           =           new
System.Windows.Threading.DispatcherTimer();
                        dispatcherTimer.Tick        +=        new
EventHandler(saveData);
                        dispatcherTimer.Interval = new TimeSpan(0,
0, 0, 0, saveInterval);
                        dispatcherTimer.Start();

                  }

                  sensor.AllFramesReady += KinectAllFramesReady;

                  button_begin.Content = "Stop";
                  SolidColorBrush        backgroundBrush        =
(SolidColorBrush)(new BrushConverter().ConvertFrom("#FFDE8080"));
                  button_begin.Background = backgroundBrush;

                  slider_frequency.IsEnabled = false;
                  button_fileBrowse.IsEnabled = false;
                  checkBox_enableDataLogging.IsEnabled = false;
            }
            else
            {
                  // Stop data collection/streaming
                  sensor.SkeletonStream.Disable();
                  sensor.DepthStream.Disable();
                  sensor.ColorStream.Disable();
```

```
                button_begin.Content = "Begin";

                SolidColorBrush        backgroundBrush        =
(SolidColorBrush)(new BrushConverter().ConvertFrom("#FFEEC680"));
                button_begin.Background = backgroundBrush;

                if (checkBox_enableDataLogging.IsChecked.Value){
                      dispatcherTimer.Stop();
                }

                slider_frequency.IsEnabled = true;
                button_fileBrowse.IsEnabled = true;
                checkBox_enableDataLogging.IsEnabled = true;

                bodyStatus.Fill              =              new
SolidColorBrush(Colors.Gray);
                faceStatus.Fill              =              new
SolidColorBrush(Colors.Gray);

                parameter_headYaw.Text = "0";
                parameter_upperBodyLean.Text = "0";
                parameter_attentionStatus.Content = "Unknown";
                parameter_attentionStatus.Foreground    =    new
SolidColorBrush(Colors.Gray);


            }

      }


}

private double getUpperBodyLeanAngle(Skeleton skel)
{
      // calculate upper body lean angle...

      var shoulderCenter = skel.Joints[JointType.ShoulderCenter];
      var hipCenter = skel.Joints[JointType.HipCenter];

      //  see  http://stackoverflow.com/questions/19729831/angle-
between-3-points-in-3d-space
      var  v1  =  new[]  {  0,  shoulderCenter.Position.Y  -
hipCenter.Position.Y,  0  };  //  this  is  the  vertical;  assume  y
component equals shoulder y component (valid for small angles)
      var    v2    =    new[]    {    shoulderCenter.Position.X    -
hipCenter.Position.X,         shoulderCenter.Position.Y         -
hipCenter.Position.Y,         shoulderCenter.Position.Z         -
hipCenter.Position.Z };

      var v1mag = Math.Sqrt(v1[0] * v1[0] + v1[1] * v1[1] + v1[2]
* v1[2]);
      var v1norm = new[] { v1[0] / v1mag, v1[1] / v1mag, v1[2] /
v1mag };
```

```
      var v2mag = Math.Sqrt(v2[0] * v2[0] + v2[1] * v2[1] + v2[2]
* v2[2]);
      var v2norm = new[] { v2[0] / v2mag, v2[1] / v2mag, v2[2] /
v2mag };

      int angleSign;
      var res = v1norm[0] * v2norm[0] + v1norm[1] * v2norm[1] +
v1norm[2] * v2norm[2];
      if  (shoulderCenter.Position.Z  <  hipCenter.Position.Z)  {
angleSign = 1; } else { angleSign = -1; }
      var angle = angleSign * Math.Round(Math.Acos(res) * (180.0 /
Math.PI));
      return angle;
}
}
}
```

List of Symbols, Abbreviations, and Acronyms

3-D	3-dimensional
API	application programming interface
CERT	computer expression recognition toolbox
CS	c sharp (C#)
CSV	comma-separated value
EEG	electroencephalogram
FACS	facial action coding system
IR	infrared
PCPT	PEBL continuous performance test
PEBL	psychology experiment building language
PPVT	PEBL perceptual vigilance task
RGB	red, green, blue
RGB-D	red, green, blue, depth
SDK	software development kit
TOAV	test of attentional vigilance
WPF	Windows presentation foundation
USB	universal serial bus
XAML	extensible application markup language

1 (PDF) DEFENSE TECHNICAL INFORMATION CTR DTIC OCA

2 (PDF) DIR ARL
RDRL CIO L
IMAL HRA MAIL & RECORDS MGMT

1 (PDF) GOVT PRINTG OFC
A MALHOTRA

1 (PDF) ARL
RDRL HRB B
T DAVIS
BLDG 5400 RM C242
REDSTONE ARSENAL AL 35898-7290

8 (PDF) ARL
SFC PAUL RAY SMITH CENTER
RDRL HRO COL H BUHL
RDRL HRF J CHEN
RDRL HRA I MARTINEZ
RDRL HRR R SOTTILARE
RDRL HRA C A RODRIGUEZ
RDRL HRA B G GOODWIN
RDRL HRA A C METEVIER
RDRL HRA D B PETTIT
12423 RESEARCH PARKWAY
ORLANDO FL 32826

1 (PDF) USA ARMY G1
DAPE HSI B KNAPP
300 ARMY PENTAGON
RM 2C489
WASHINGTON DC 20310-0300

1 (PDF) USAF 711 HPW
711 HPW/RH K GEISS
2698 G ST BLDG 190
WRIGHT PATTERSON AFB OH 45433-7604

1 (PDF) USN ONR
ONR CODE 341 J TANGNEY
875 N RANDOLPH STREET
BLDG 87
ARLINGTON VA 22203-1986

1 (PDF) USA NSRDEC
RDNS D D TAMILIO
10 GENERAL GREENE AVE
NATICK MA 01760-2642

1 (PDF) OSD OUSD ATL
HPT&B B PETRO
4800 MARK CENTER DRIVE
SUITE 17E08
ALEXANDRIA VA 22350

ABERDEEN PROVING GROUND

12 (PDF) ARL
RDRL HR
J LOCKETT
P FRANASZCZUK
K MCDOWELL
K OIE
RDRL HRA AA
C GARNEAU
RDRL HRB
D HEADLEY
RDRL HRB C
J GRYNOVICKI
RDRL HRB D
C PAULILLO
RDRL HRF A
A DECOSTANZA
RDRL HRF B
A EVANS
RDRL HRF C
J GASTON
RDRL HRF D
A MARATHE